LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

LA-UR-87-2187

LA-UR--87-2187

DE87 011735

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

TITLE: USING THE NEWS WINDOW SYSTEM IN A CRAY ENVIRONMENT

AUTHOR(S): R. L. Phillips
D. W. Forslund

SUBMITTED TO: Cray User Group, proceedings for Spring 1987 meeting.

DISCLAIMER

MASTER

Los Alamos Los Alamos National Laboratory
Los Alamos, New Mexico 87545

USING THE NeWS WINDOW SYSTEM IN A CRAY ENVIRONMENT

R. L. Phillips
D. W. Forslund

Los Alamos National Laboratory
Los Alamos, NM

INTRODUCTION

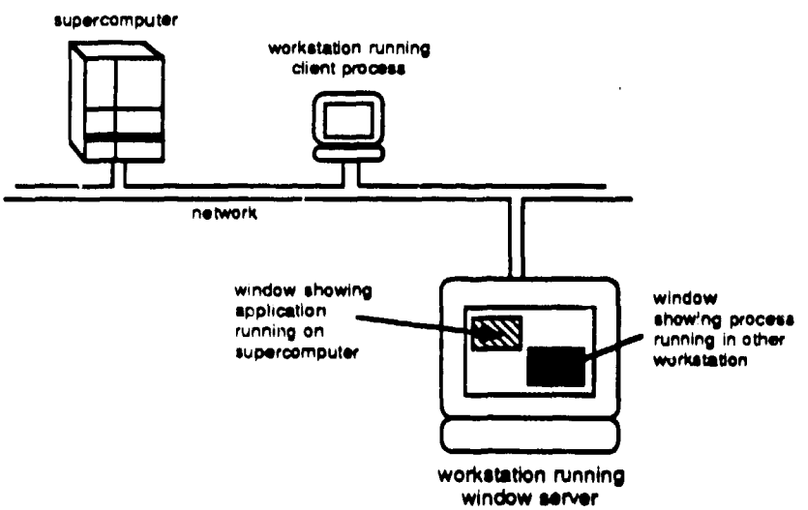
The computing environment at Los Alamos National Laboratory (LANL) can be characterized as being widely distributed and massively interconnected. Workstations, terminals, and mainframes can all communicate with one another with relative ease. Although large worker machines, such as the eight Crays at LANL, can be accessed interactively through conventional command line driven interfaces, often applications are best controlled and analyzed via a graphical user interface. Workstations, such as those made by Sun Microsystems, Inc. and Apollo Computer Inc., provide the high resolution displays and mouse-driven graphical input needed for such a user interface, but they lack the computing power required for many applications at LANL.

What is needed is a distribution of processing, so that the main computational burden is borne by a Cray while the interaction tasks are handled by the workstation. NeWS, a product of Sun Microsystems, is a distributed, extensible window system that provides for easy allocation of computing tasks throughout a distributed environment. NeWS differs from other distributed window systems in that the messages sent between processes are in the form of PostScript programs. This communication mechanism promotes a high degree of extensibility and device independence.

NeWS is structured as a single UNIX process, a network server that contains a PostScript interpreter. Client programs, which exist somewhere out on the network, talk to NeWS through byte streams. This paper describes the implementation of such a client interface on a Cray running the UNICOS operating system. With only a modest effort, it is possible to fit a simple PostScript interface to existing mainframe applications, which allows the user to graphically interact with the program from a remote workstation. Some typical applications, which have been structured as NeWS clients, will be described.

NeWS OVERVIEW

The acronym NeWS, derived from Network-extensible Window System, suggests the network orientation of this window system. As such, it allows the computing resources in a heterogeneous computing environment to be effectively used. NeWS runs on a machine with one or more bitmapped displays. It acts as a *window server*, managing input and output on its host machine. Application programs — called *clients* — send messages that cause NeWS to render images on the display. The clients may reside anywhere on the network. Server-based window systems are often called *distributed window systems*, because the server and its clients may be distributed over the network. This concept is not unique to NeWS. It first appeared in Andrew,[1] a system developed at Carnegie-Mellon, and X Window,[2] from MIT. The following diagram depicts this network orientation, where the NeWS server, running on a workstation, serves remote clients running on other machines.



One of the key features of NeWS is the use of a programming language for communication between the server and its clients. Instead of sending messages consisting of commands and parameters in a fixed format, clients send *programs* that the server interprets. The language used by NeWS is PostScript,[3] which was originally developed as a language to drive printers. In order to be suitable for use in an interactive, workstation—based environment, Sun has augmented the basic PostScript language with new data types and operators. Among the most significant of these enhancements are *canvases* and *events*. Whereas a printer deals with but one page at a time, NeWS can write to multiple display

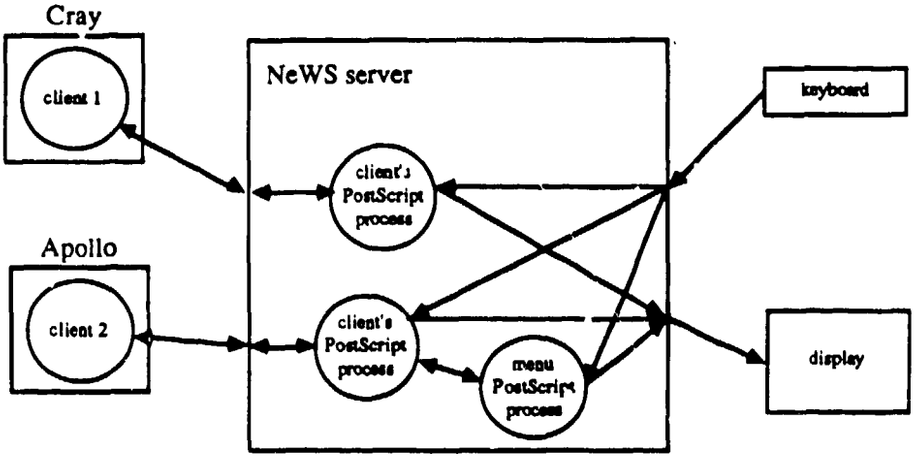
[1] J. Morris et. al., "Andrew: A Distributed Personal Computing Environment," Communications of the ACM, Vol. 29, No. 3 (March 1986).

[2] James Gettys, "Problems Implementing Window Systems in UNIX," *USENIX Proceedings*, January 1986.

[3] Adobe Systems, Inc., *PostScript Language Reference Manual*, Addison-Wesley, 1985.

surfaces, or canvases. PostScript processes (clients) draw on canvases using PostScript graphics primitives. Canvases are cheap and fast to create and NeWs makes liberal use of them. Anything drawn on the screen uses a canvas, whether it is a transient menu or a semi-permanent window where an application's data are displayed.

Although a PostScript—equipped printer has no need ɔ deal with interactive input from a keyboard or a mouse, a workstation running NeWS must be able to deal with these devices. For this, Sun added the *event* data type to PostScript. Events are PostScript objects that can be generated either by PostScript processes or by external devices such as the mouse and keyboard. A process can send an event to itself or any other other process. Events can be directed to specific processes or canvases by filling in the appropriate field in the event's data structure. The following diagram shows a possible scenario for communication paths between the server and various clients and devices in a network environment.



Other enhancements that NeWS brings to PostScript, such as lightweight processes and objects for describing color, graphics state, current path (shape), and interprocess communication, while important, are beyond the scope of this paper.[4]

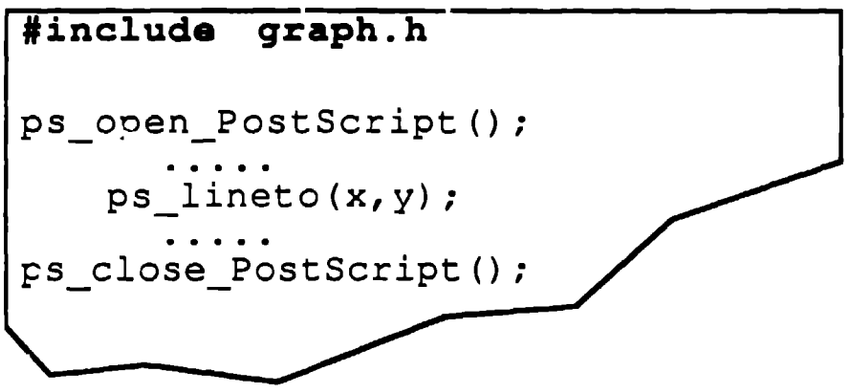
CLIENT/SERVER COMMUNICATION

The programmer can deal with NeWS at various levels but the most common is one where a bridge is built between a new or existing C language[5] application and the PostScript environment of the server. The programmer does this by writing a specification file that

[4] Sun Microsystems, Inc., *NeWS Technical Overview*, Part No. 800-1498-05, March, 1987.

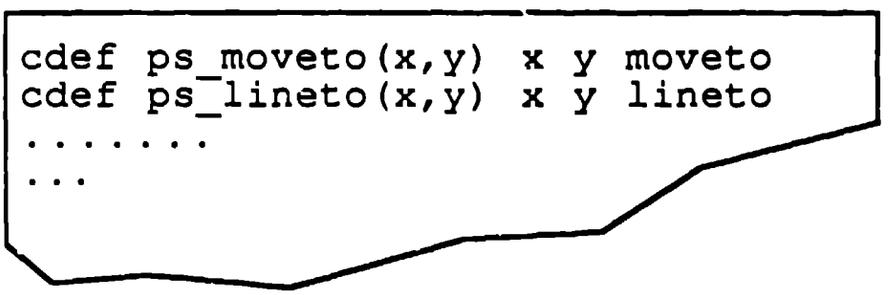
[5] NeWS is not restricted to programs written in C but that is the only language binding currently available.

associates C procedure names with PostScript code that is to be sent to the server when the C procedure is invoked by a client. This specification file is compiled by a program called CPS (for "C to PostScript") into a C header file, which is included by the C application. This situation is depicted in the following diagram.



application program (client)

cps graph.cps > graph.h



Interface program (PostScript)

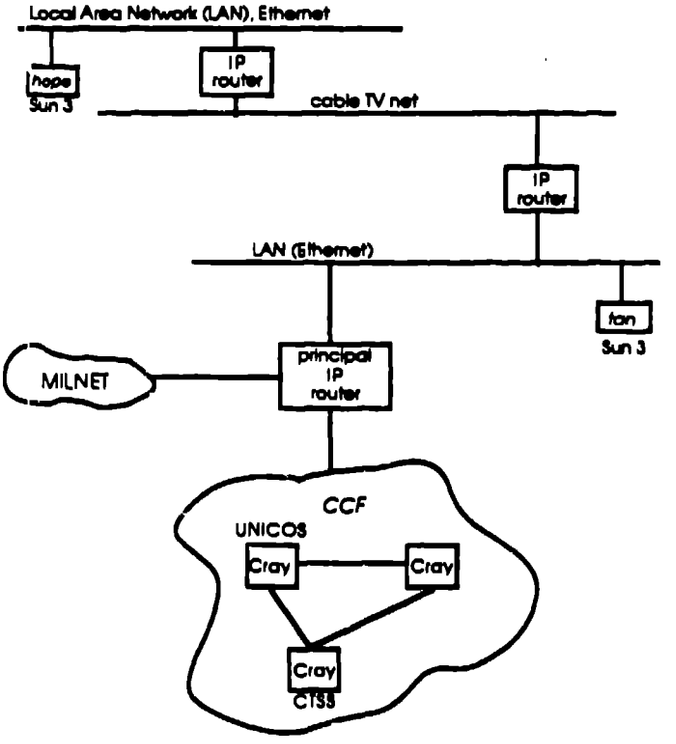
This example suggests that an application will draw lines in a window on the workstation display by invoking a procedure called *ps_lineto* with arguments x and y. Any name could have been chosen as long as it is unique. Prior to calling that procedure, however, the application must invoke a standard library routine supplied with NeWS, which opens a path between the client (application) and the server, *ps_open_PostScript*. When finished, the application closes that path by calling *ps_close_PostScript*.

Next, meaning is given to the user-defined procedure by specifying the relationship between the procedure and corresponding PostScript commands. This step is indicated in the second part of the diagram. The directive *cdef* specifies that the C procedure name that follows, when invoked, is to send the ensuing PostScript commands to the NeWS server.

Here, *ps_lineto(x,y)* is equivalent to the PostScript statement *x y lineto*. This is a trivial example[6] but it does demonstrate the ease with which the C and PostScript worlds can be bridged. NeWS provides many other tools to facilitate communication between an application and the server. Some of these will be described in the section **Enhancing an Application: A Case Study.**

PORTING NeWS TO A CRAY

Earlier it was mentioned that NeWS is a network—oriented window system and that clients and servers can exist anywhere in the network. A situation that is especially useful is one where a computationally burdensome application (client) is running on a supercomputer and the server, as is usually the case, is running on a Sun workstation.[7] In this case the supercomputer is a Cray X-MP/24, which is accessible over a network that uses TCP/IP[8] protocols. A segment of the pertinent network topology is shown in the following diagram.



[6] In fact, most of the common PostScript graphics operations, like *moveto* and *lineto*, are predefined.

[7] Ports of the NeWS server have been done for Apollo workstations and for some less powerful graphics-based personal computers as well.

[8] This is a standard Defense Advanced Research Projects Agency (DARPA) internet protocol. The acronym derives from Transmission Control Protocol/Internet Protocol.

The boxes labeled *IP router* are gateways between different physical networks. The Sun workstations named *tan* and *hope* are the names of the machines used by the authors. They are some 2 miles distant from one another and both access the Cray located in the Central Computational Facility through the gateway labeled *principal IP router*.

Generally, it is a straightforward matter to port the necessary client software to a new machine if there is a UNIX environment available. Cray provides UNICOS, which is based on AT&T System V UNIX. The NeWS client software provided by Sun is 4.2 BSD based, which results in a few minor incompatibilities with UNICOS. Besides the client application program, one must port the CPS compiler and a related library. This was accomplished with little effort once a C compiler that supports symbol names longer than eight characters was provided. A final porting issue arose because, for performance reasons, NeWS compresses its PostScript data stream to an encoded form. There were a few problems encountered in ensuring that these *tokens*, as they are called, would propagate unmodified over the network.

Once the port was accomplished, access to the client application of the Cray was gained by opening a terminal window on a workstation running the NeWS server and executing a remote login to the Cray. Once established, one simply invokes the application and it establishes contact with the server via the *ps_open_PostScript* procedure described earlier. All keyboard and mouse interaction required by the application is handled by the workstation. All output from the client is displayed on the workstation screen.

ENHANCING AN APPLICATION: A CASE STUDY

NeWS is basically a window server rather than a complete window system. This means that NeWS does not presume a window or menu management strategy or enforce a particular style of user interface. To demonstrate the capabilities of NeWS, however, Sun provides a set of files that implement a sample user interface, a window system, and a menu package. For ease of use, and to minimize the programmer's requisite knowledge of PostScript, these utilities have been provided in the style of Smalltalk-like objects.[9] Generally, a programmer will find it convenient to use these tools when adapting a client application to the NeWS environment. This is disarmingly easy to do, as is shown in the following program fragments.

[9] Owen M. Densmore, "Object Oriented Programming in NeWS," *Third Monterey Graphics Workshop, USENIX*, November, 1986.

```
/win framebuffer /new DefaultWindow send def
{
   /FrameLabel (Hello!) def
   /PaintClient {paintme} def
   /IconLabel (TEST) def
   /ClientMenu MyMenu def
}  win send
100 100 200 300 /reshape win send
/map win send
```

This is an example of how one creates an instance of a window object. *DefaultWindow* is the name of the master window class, i.e., the sample window manager supplied with NeWS. The identifier */win* is arbitrarily chosen and will contain the definition of a window whose parent is *framebuffer*, the primary display, and is created by the window class method called */new*. Then four procedures are installed in *win* which label the frame (*FrameLabel*), produce output from the client in the window (*PaintClient*), and identify the window in the closed, iconic state (*IconLabel*). The next line specifies that a specific menu is to be installed for this window, *MyMenu*, which is defined below. Finally, the window is sized to appear with its lower left corner at 100,100 units, with a width of 200 and a height of 300 units. The window is made visible by sending the object the */map* method.

Menu creation is equally straightforward. The following is a typical command for producing a menu of three items.

```
/MyMenu
[
   (Key 1)   {menuproc1}
   (Key 2)   {menuproc2}
   (Other =>) OtherMenu
]  /new DefaultMenu send def
```

This statement creates an instance of the master menu class *DefaultMenu* called *MyMenu*. When an item named *Key 1* is selected it causes a procedure named *menuproc1* to be invoked. Likewise for the second item. The third item, *Other* =>, causes a pull-right or walking menu to appear, *OtherMenu*, which has a similar set of keys and corresponding action procedures. *MyMenu* appears when a specified mouse button is pressed when the cursor is within the window to which it belongs — *win* in this case.

The first sample application to be considered is GNUPLOT, a public domain program developed at Villanova University.[10] GNUPLOT is a sophisticated interactive data plotting program comprising several thousand lines of C source code. About 30 lines of code were added to provide the client/server bridge described earlier. Four C-callable procedures were defined in terms of about 400 lines of PostScript. Once processed with CPS, the resulting header file is included and compiled with the slightly modified GNUPLOT application. The default master window and menu classes were used without modification. The behavior of GNUPLOT as modified to run under NeWS can best be described by reference to a sequence of screen dumps, Plates 1 - 4.

Plate 1 shows three VT100 terminal emulator windows. GNUPLOT has been invoked in the upper left window and commands to the program will be issued there. A collection of typical GNUPLOT commands is shown in the lower right window. Finally, a portion of the PostScript interface program is shown in the lower left window. In particular, the complete routine for drawing in a window, *PaintClient*, is shown there. In the upper left window a command has been issued to GNUPLOT to plot the sine function. The small window to the right was opened and the data were scaled and plotted there. It is important to note that the client application (which, recall, resides on a Cray) need only be consulted when a new data set is requested. Otherwise, all operations are performed locally on the workstation that hosts the NeWS server. The menu[11] appearing at the bottom of the plotting window shows that five operations can now be performed. These are all handled by the PostScript interface; GNUPLOT knows nothing of these capabilities.

The **New Window** capability is demonstrated in Plate 2. There, three additional windows have been interactively opened by the user. Several commands were issued to GNUPLOT, resulting in the new data sets being plotted in the additional windows. The **Active Window** menu item allows the user to steer a new data set to the desired window,thereby to draw afresh in empty ones or to replace data sets in those windows currently in use.

Plate 3 demonstrates the **Zoom** menu item. By invoking it the user has zoomed-in to the upper left corner of the original sine curve data set (in the window labeled *GNUPLOT0*), which reveals the label that was previously illegible. A new menu item, **Restore**, has also appeared, which allows restoration of the original scale. After zooming, the user invoked the **Print** menu item, which produced a file containing the PostScript representation of the current window contents ready for printing. The window to the left, labeled *psview*, demonstrates the application of a NeWS PostScript previewing facility to that file.

[10] GNUPLOT was developed by Thomas Williams and Colin Kelley, Department of Electrical Engineering, Villanova University, Villanova, PA 19085.

[11] *ClientMenu*, to use previous terminology.

Finally, Plate 4 shows an application of the **Overlay** menu item. This is designed to allow comparison of different data sets. The tangent function originally plotted in the *GNUPLOT1* window was overlaid on the sine function in *GNUPLOT0*. The same operation was performed in *GNUPLOT1* and the user zoomed-in on the origin of the composite data set. When the **Print** menu item is invoked for an overlaid window, the composite, possibly **Zoomed**, data set will be printed.

OTHER CLIENT APPLICATIONS

Two other existing applications have been converted to NeWS clients, using the same client/server interface techniques described earlier. As was the case with GNUPLOT, only minimal changes had to be added to the applications to provide access to NeWS capabilities. The first of these applications is SURFMODL,[12] a public domain surface modeling package. Plate 5 shows some images produced by that package. Consistent with the complexity of the application, its menu is more extensive than that for GNUPLOT. Note, however, that the first menu **Window Edit** is, when invoked, the entire GNUPLOT menu, *unchanged*. Thus, it was possible to reuse all the functionality that was developed for another application. Plate 6 is a good example of the use of the **Print** menu command. It depicts a shaded rendition of the image shown in the *SURFMODL0* window.

Plate 7 shows shows an application where, again, the menu functions developed for GNUPLOT were used virtually unchanged. This application is a standard UNIX plot file filter. Plot files conforming to this format can be previewed, independent of the application that produced them. In this example a file containing four plots is being previewed. The data plotted in the window labeled *NeWSplot1* were extracted from the original window with the **Zoom** command. The ability to leverage off previous NeWS developments should be obvious.

CONCLUDING REMARKS

NeWS has proven to be a flexible and powerful environment for building sophisticated user interfaces for major applications. Particularly impressive is the ease with which existing applications can be fitted with an interactive interface, even though issues like window systems and graphical input were never envisioned in their design. This is especially important in an organization where many substantial applications already exist and it is not feasible to change them radically in order to accommodate a workstation

[12] SURFMODL was developed by Kenneth Van Camp, P. O. Box 784, Stroudsburg, PA 18360.

environment. The authors have just begun to explore the potential of NeWS. As impressive as the results have been so far, it seems the surface has hardly been scratched.

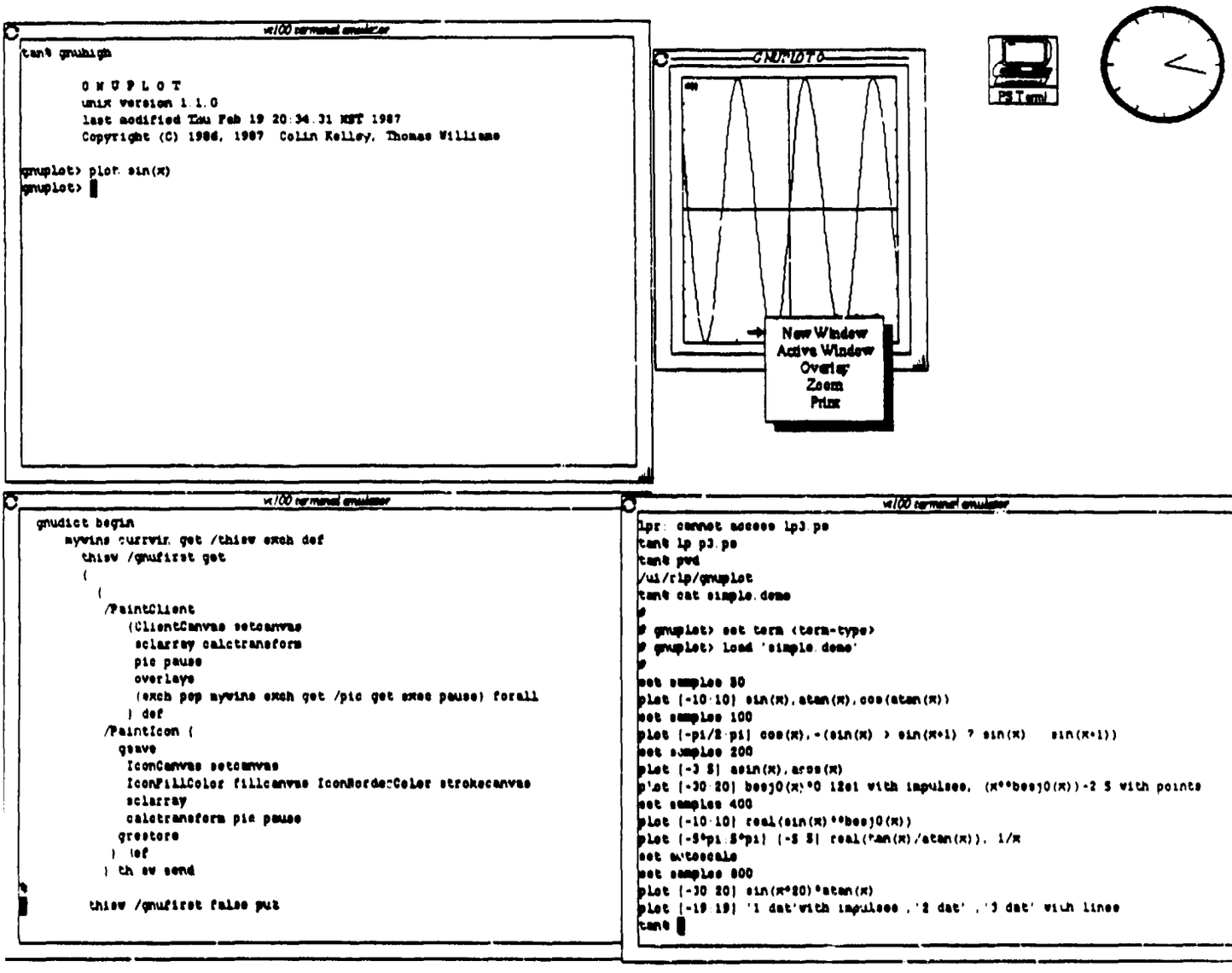

w100 terminal emulator
tank gnuhigh
G N U P L O T
unix version 1.1.0
last modified Thu Feb 19 20:34.31 MST 1987
Copyright (C) 1986, 1987 Colin Kelley, Thomas Williams
gnuplot> plot sin(x)
gnuplot>
GNUPLOT
New Window
Active Window
Overlay
Zoom
Print
PS Term
gnudict begin
mywins currvin get /thisw exch def
thisw /gnufirst get
/PaintClient
{ClientCanvas setcanvas
sclarray calctransform
pic pause
overlays
{exch pop mywins exch get /pic get exec pause} forall
} def
/PaintIcon {
gsave
IconCanvas setcanvas
IconFillColor fillcanvas IconBorderColor strokecanvas
sclarray
calctransform pic pause
grestore
} def
} th sw send
thisw /gnufirst false put
w100 terminal emulator
lpr: cannot access lp3.ps
tank lp p3.ps
tank pwd
/ui/rlp/gnuplot
tank cat simple.demo
#
gnuplot> set term {term-type}
gnuplot> load 'simple.demo'
#
set samples 50
plot [-10:10] sin(x),atan(x),cos(atan(x))
set samples 100
plot [-pi/2:pi] cos(x),-(sin(x) > sin(x+1) ? sin(x) : sin(x+1))
set samples 200
plot [-3:3] asin(x),acos(x)
plot [-30:20] besj0(x*0.12e1 with impulses, (x**besj0(x))-2.5 with points
set samples 400
plot [-10:10] real(sin(x)**besj0(x))
plot [-5*pi:5*pi] [-5:5] real(tan(x)/atan(x)), 1/x
set autoscale
set samples 800
plot [-30:20] sin(x*20)*atan(x)
plot [-19:19] '1.dat'with impulses ,'2.dat' ,'3.dat' with lines
tank

Plate 1

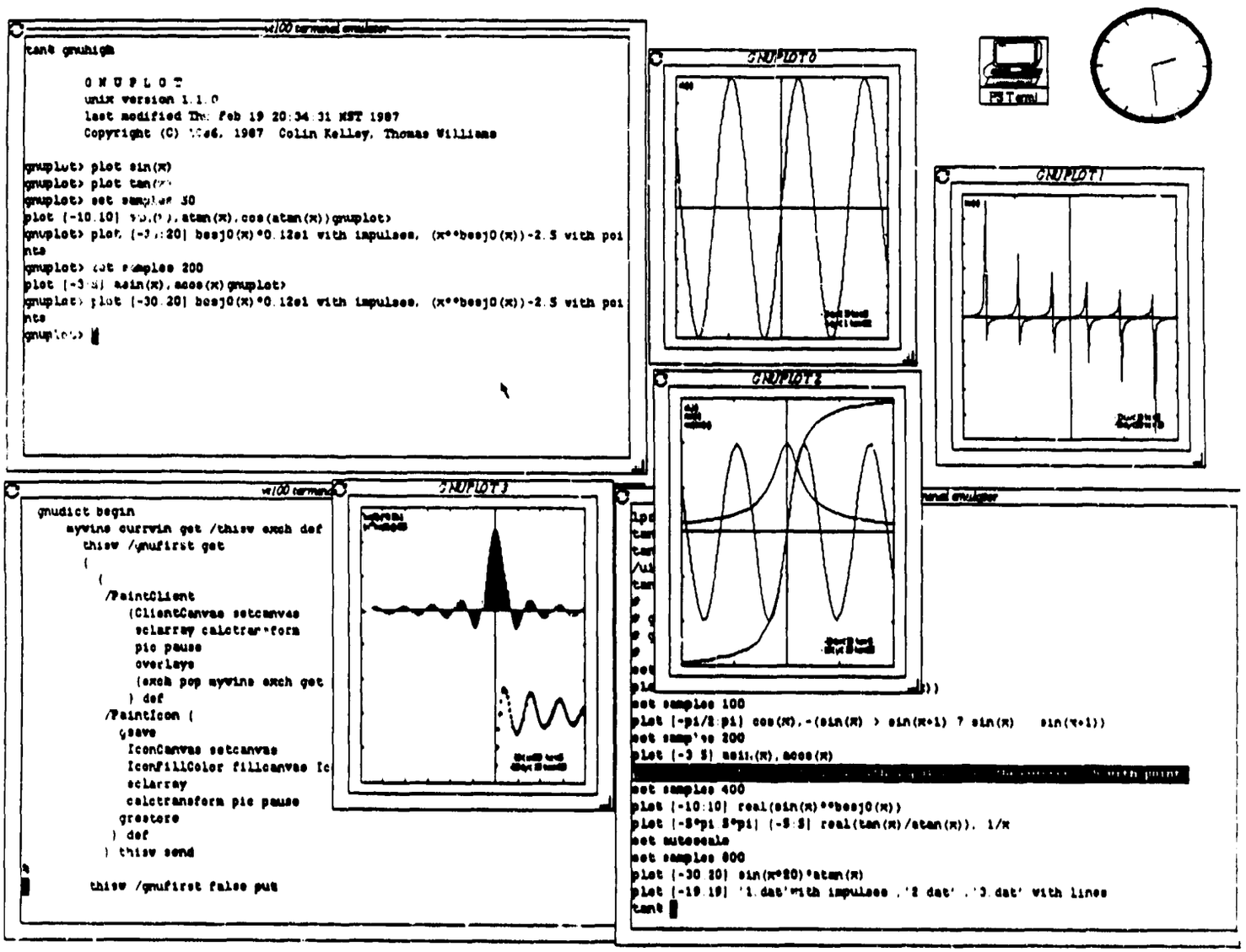

GNUPLOT0
GNUPLOT1
GNUPLOT2
GNUPLOT3
PS Terml
tan% gnuhigh
G N U P L O T
unix version 1.1.0
Copyright (C) 1986, 1987 Colin Kelley, Thomas Williams
gnuplot> plot sin(x)
gnuplot> set samples 50
gnuplot> set samples 200
gnuplot> plot [-3:3] asin(x),acos(x)gnuplot>
gnuplot> plot [-30:20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2.5 with points
gnuplot>
gnudict begin
/PaintClient
/PaintIcon {
thisw /gnufirst false put
set samples 100
plot [-pi/2:pi] cos(x),-(sin(x) > sin(x+1) ? sin(x) : sin(x+1))
set samples 200
plot [-3:3] asin(x),acos(x)
set samples 400
plot [-10:10] real(sin(x)**besj0(x))
plot [-3*pi:3*pi] [-3:3] real(tan(x)/atan(x)), 1/x
set autoscale
set samples 800
plot [-30:20] sin(x*20)*atan(x)
plot [-10:10] '1.dat' with impulses ,'2.dat' ,'3.dat' with lines

Plate 2

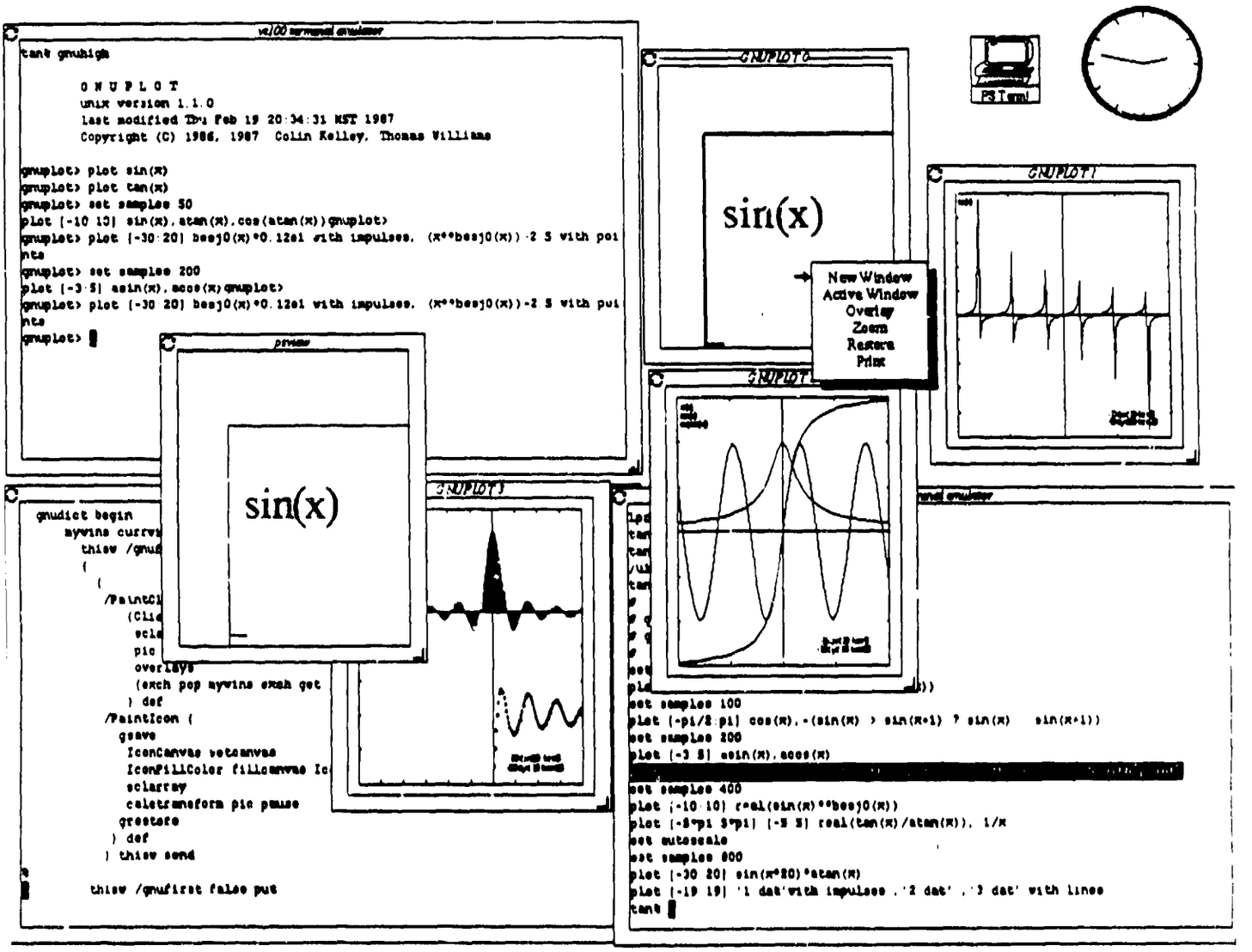

tan% gnuhigh
G N U P L O T
Unix version 1.1.0
last modified Thu Feb 19 20:34:31 MST 1987
Copyright (C) 1986, 1987 Colin Kelley, Thomas Williams
gnuplot> plot sin(x)
gnuplot> plot tan(x)
gnuplot> set samples 50
plot [-10 10] sin(x),atan(x),cos(atan(x))gnuplot>
gnuplot> plot [-30:20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2 5 with poi
nts
gnuplot> set samples 200
plot [-3 5] asin(x),acos(x)gnuplot>
gnuplot> plot [-30 20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2 5 with pui
nts
gnuplot>
GNUPLOT0
sin(x)
New Window
Active Window
Overlay
Zoom
Restore
Print
GNUPLOT1
preview
sin(x)
GNUPLOT3
gnudict begin
thisw /gnufirst false put
set samples 100
plot [-pi/2:pi] cos(x),-(sin(x) > sin(x+1) ? sin(x) : sin(x+1))
set samples 200
plot [-3 5] asin(x),acos(x)
set samples 400
plot [-10:10] real(sin(x)**besj0(x))
plot [-8*pi:8*pi] [-5 5] real(tan(x)/atan(x)), 1/x
set autoscale
set samples 800
plot [-30 20] sin(x*20)*atan(x)
plot [-19 19] '1.dat'with impulses ,'2.dat' ,'3.dat' with lines
tan%

Plate 3

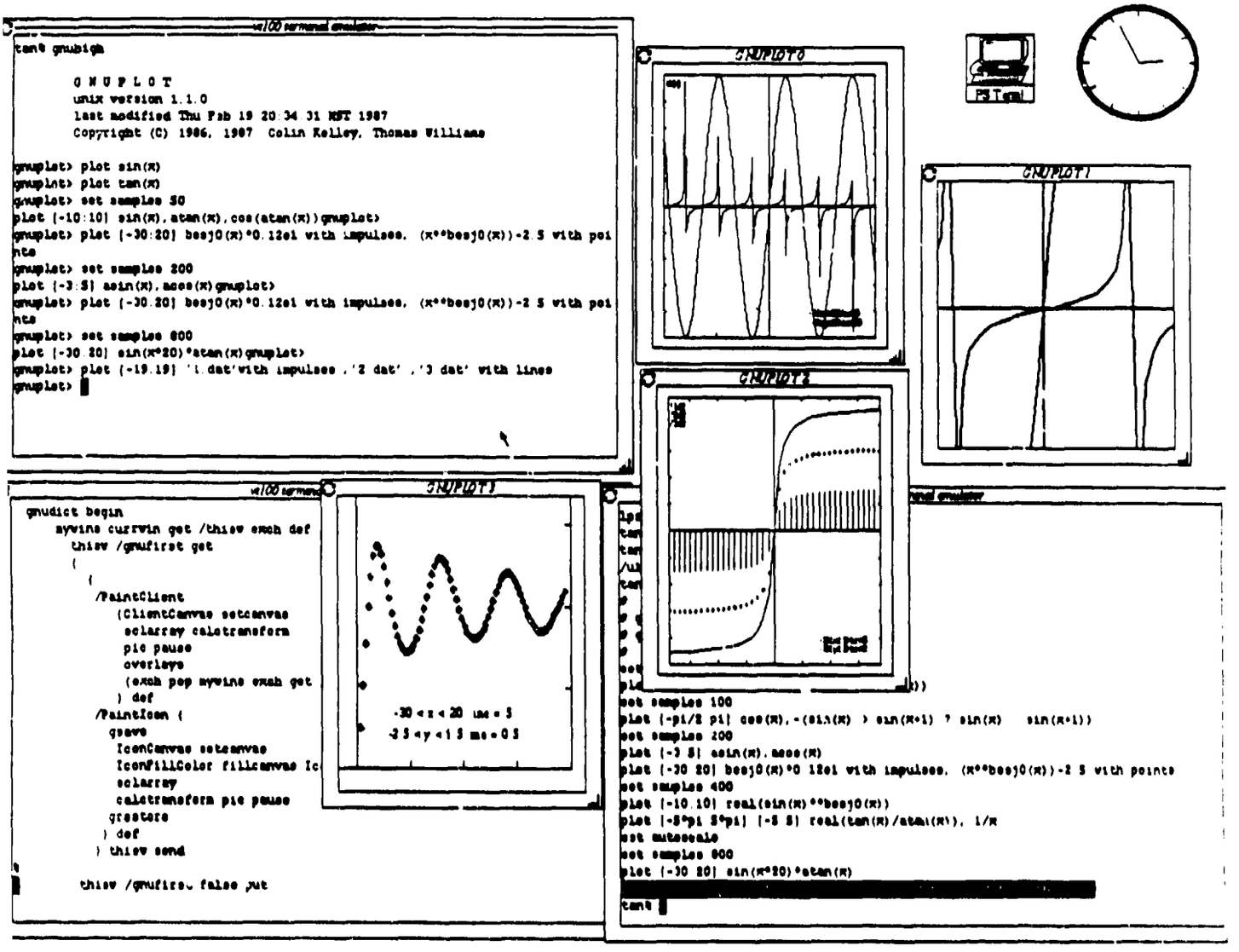

tan% gnuplot
G N U P L O T
unix version 1.1.0
last modified Thu Feb 19 20:34:31 MST 1987
Copyright (C) 1986, 1987 Colin Kelley, Thomas Williams
gnuplot> plot sin(x)
gnuplot> plot tan(x)
gnuplot> set samples 50
plot [-10:10] sin(x),atan(x),cos(atan(x))gnuplot>
gnuplot> plot [-30:20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2.5 with points
gnuplot> set samples 200
plot [-3:5] asin(x),acos(x)gnuplot>
gnuplot> plot [-30:20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2.5 with points
gnuplot> set samples 800
plot [-30:20] sin(x*20)*atan(x)gnuplot>
gnuplot> plot [-19:19] '1.dat'with impulses ,'2 dat' ,'3 dat' with lines
gnuplot>
GNUPLOT0
GNUPLOT1
GNUPLOT2
GNUPLOT3
-30 < x < 20 inc = 5
-2.5 < y < 1.5 inc = 0.5
gnudict begin
/mywins currwin get /thisw exch def
thisw /gnufirst get
/PaintClient
(ClientCanvas setcanvas
sclarray calctransform
pic pause
overleye
(exch pop mywins exch get
) def
/PaintIcon {
gsave
IconCanvas setcanvas
IconFillColor fillcanvas Ic
sclarray
calctransform pic pause
grestore
} def
} thisw send
thisw /gnufirst false put
set samples 100
plot [-pi/2 pi] cos(x),-(sin(x) > sin(x+1) ? sin(x) : sin(x+1))
set samples 200
plot [-3.5] asin(x),acos(x)
plot [-30 20] besj0(x)*0.12e1 with impulses, (x**besj0(x))-2.5 with points
set samples 400
plot [-10:10] real(sin(x)**besj0(x))
plot [-5*pi 5*pi] [-5 5] real(tan(x)/atan(x)), 1/x
set autoscale
set samples 800
plot [-30 20] sin(x*20)*atan(x)
tan%

Plate 4

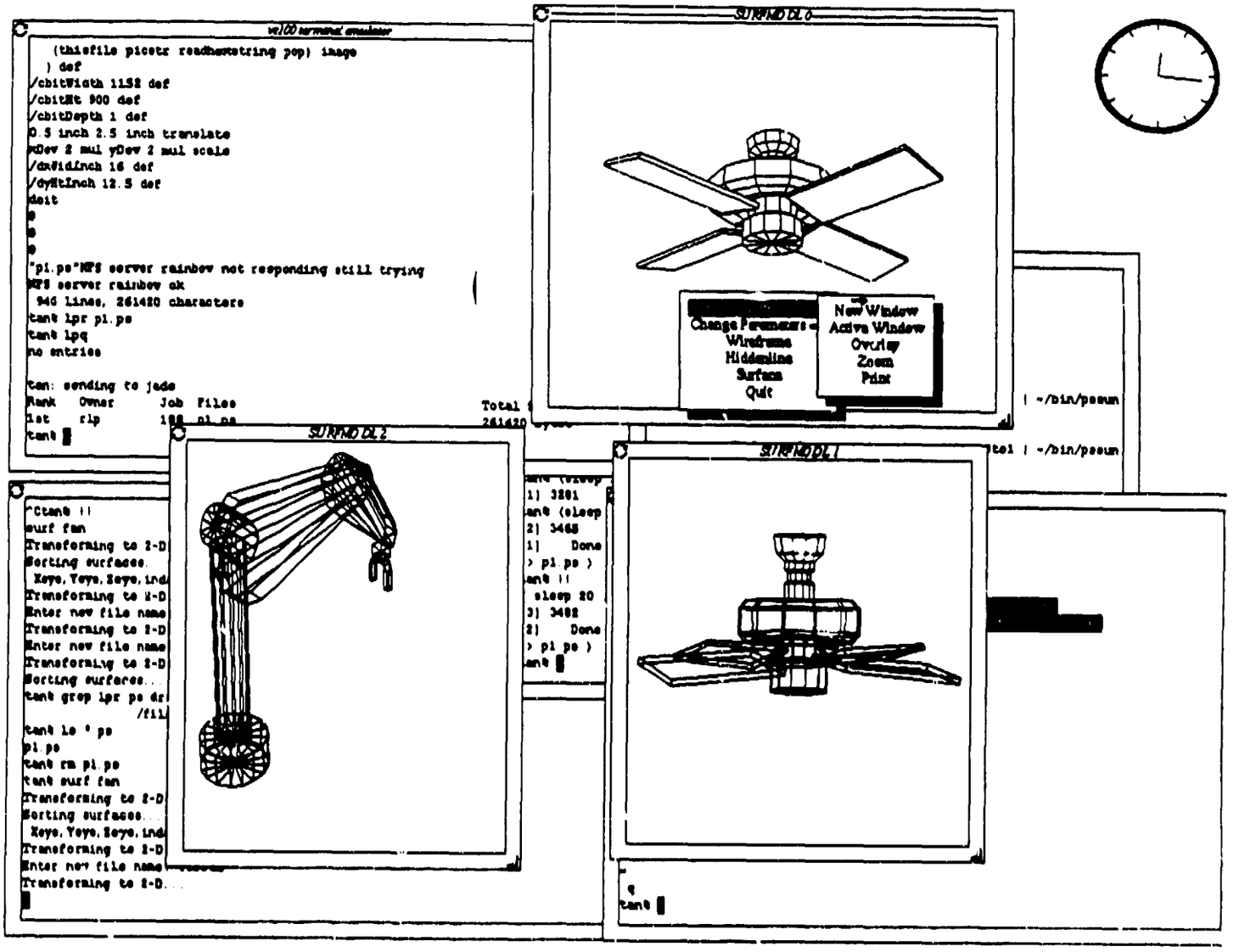

SURFMODL 0
Change Parameters
Wireframe
Hiddenline
Surface
Quit
New Window
Active Window
Overlap
Zoom
Print
SURFMODL 2
SURFMODL 1

Plate 5

Plate 6

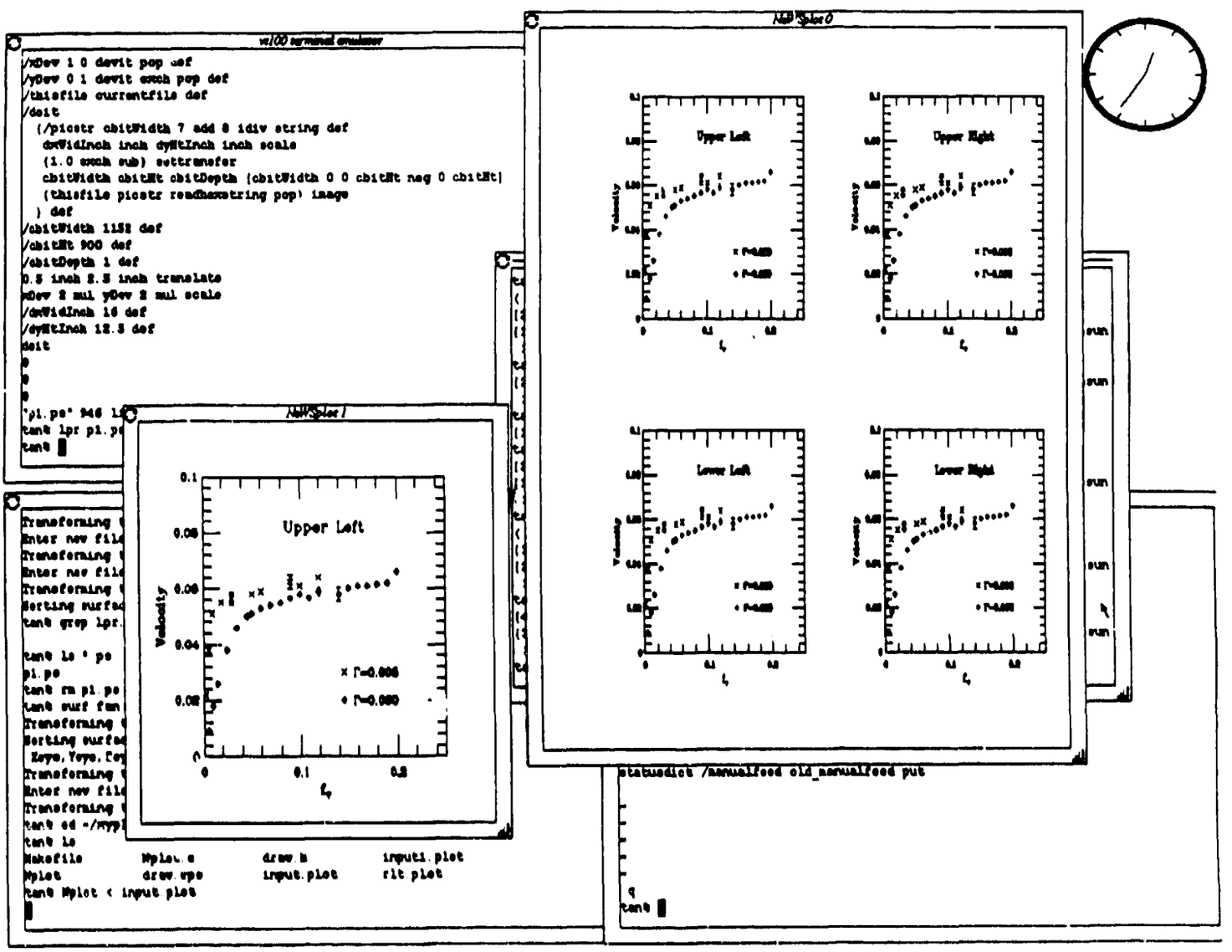

Upper Left
Upper Right
Lower Left
Lower Right
Velocity
Upper Left
Velocity
0.1
0.08
0.06
0.04
0.02
0
0.1
0.2
x Γ=0.006
• Γ=0.000
/xDev 1 0 devit pop def
/yDev 0 1 devit exch pop def
/thisfile currentfile def
/doit
/picstr cbitWidth 7 add 8 idiv string def
dxWidInch inch dyHtInch inch scale
(1.0 exch sub) settransfer
cbitWidth cbitHt cbitDepth [cbitWidth 0 0 cbitHt neg 0 cbitHt]
(thisfile picstr readhexstring pop) image
) def
/cbitWidth 1152 def
/cbitHt 900 def
/cbitDepth 1 def
0.5 inch 8.5 inch translate
xDev 2 mul yDev 2 mul scale
/dxWidInch 16 def
/dyHtInch 12.5 def
doit
Transforming
Enter new file
Transforming
Enter new file
Transforming
Sorting surfac
tank grep lpr
tank ls * ps
pl.ps
tank rm pl.ps
tank surf fan
Transforming
Sorting surfac
Xeye, Yeye, Zey
Transforming
Enter new file
Transforming
tank cd ~/myp
tank ls
Makefile Mplot.c draw.h inputi.plot
Mplot draw.vpe input.plot rlt.plot
tank Mplot < input.plot
statusdict /manualfeed old_manualfeed put
tank

Plate 7